# Ordenação

Professores:
Fátima L. S. Nunes

# Arrays

- O que é?

# Arrays

- O que é?
  - Vetor que guarda variáveis do mesmo tipo com um mesmo nome.
  - Cada posição tem um índice.

# Arrays

- Para quê?
  - Praticidade quando se deseja guardar conjuntos de valores do mesmo tipo:
    - Código mais enxuto
- Que operações são comuns em arrays?

# Arrays

- Para quê?
  - Praticidade quando se deseja guardar conjuntos de valores do mesmo tipo:
    - Código mais enxuto
- Que operações são comuns em arrays?
  - Inicializar
  - Percorrer
  - **Buscar**

# Como podemos ordenar os elementos de um array?

- Alguém joga baralho?


- Como ordenam as cartas?

# Como podemos ordenar os elementos de um array?

- Vamos incorporar duas restrições:

1. Você pode olhar no máximo 2 cartas ao mesmo tempo (mas pode anotar a posição onde está uma carta do seu interesse)
2. Você não pode mudar as cartas livremente,mas somente trocar duas cartas de posição.

# Como podemos ordenar os elementos de um array?

- Vamos incorporar duas restrições:

1. Você pode olhar no máximo 2 cartas ao mesmo tempo (mas pode anotar a posição onde está uma carta do seu interesse)

2. Você não pode mudar as cartas livremente,mas somente trocar duas cartas de posição.

Vamos tentar ordenar este array considerando essas restrições?

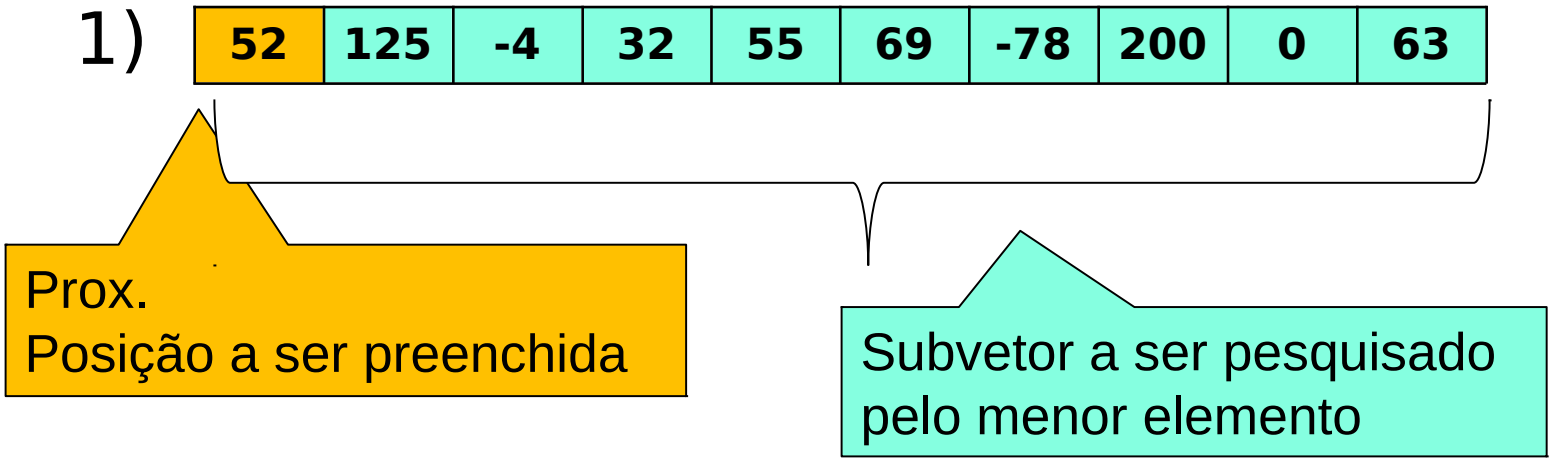
| 52 | 125 | -4 | 32 | 55 | 69 | -78 | 200 | 0 | 63 |
|---|---|---|---|---|---|---|---|---|---|

# Primeiro método: seleção direta

- Primeiro passo:
  - Encontrar o menor elemento do array
  - Levar este menor elemento para o início do array
- Segundo passo:
  - Encontrar o segundo menor elemento do array
  - Levar este menor elemento para a segunda posição do array
- E assim por diante...

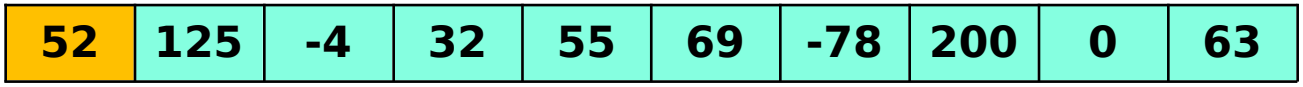
| 52 | 125 | -4 | 32 | 55 | 69 | -78 | 200 | 0 | 63 |
|---|---|---|---|---|---|---|---|---|---|

# Primeiro método: seleção direta



# Primeiro método: seleção direta

1)



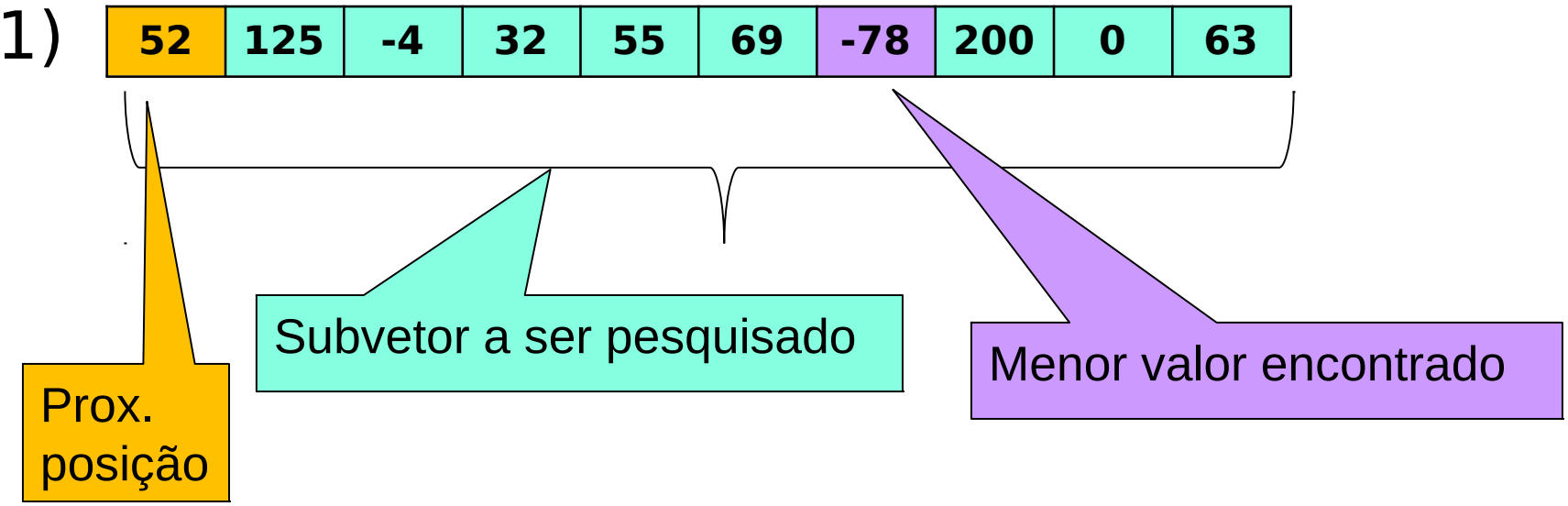

Prox.
Posição a ser preenchida

Subvetor a ser pesquisado pelo menor elemento

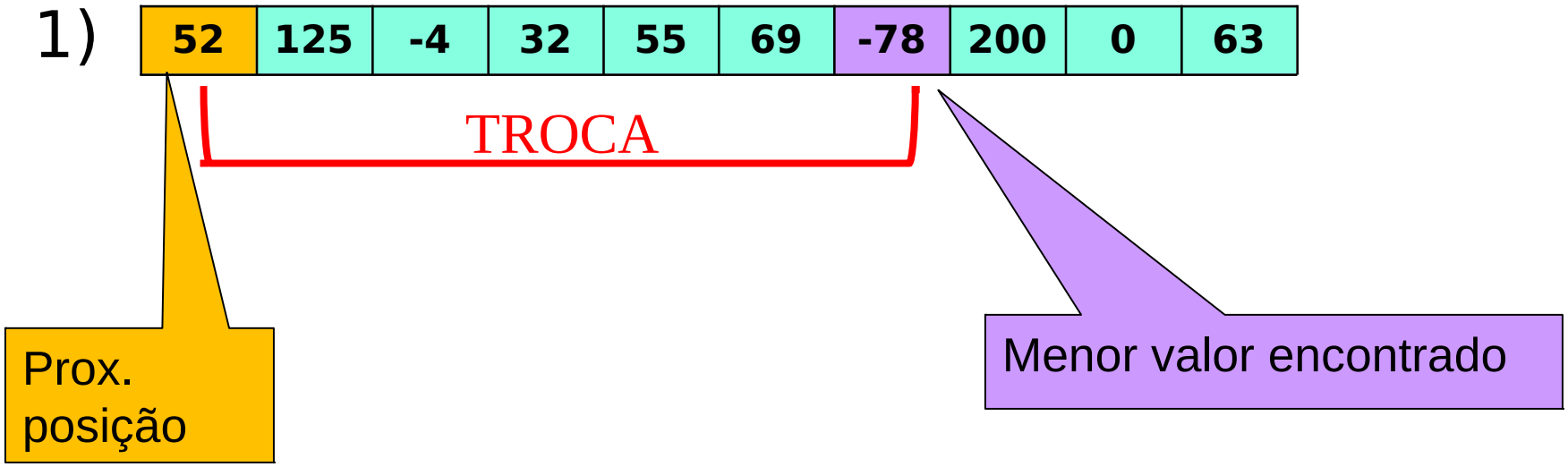
```
// código para achar o menor elemento
int iMenor = 0; // posição inicial do vetor que está buscando
for (i=1; i < N; i++){
        if (num[i] < num[iMenor])
                iMenor = i;
}
```

# Primeiro método: seleção direta

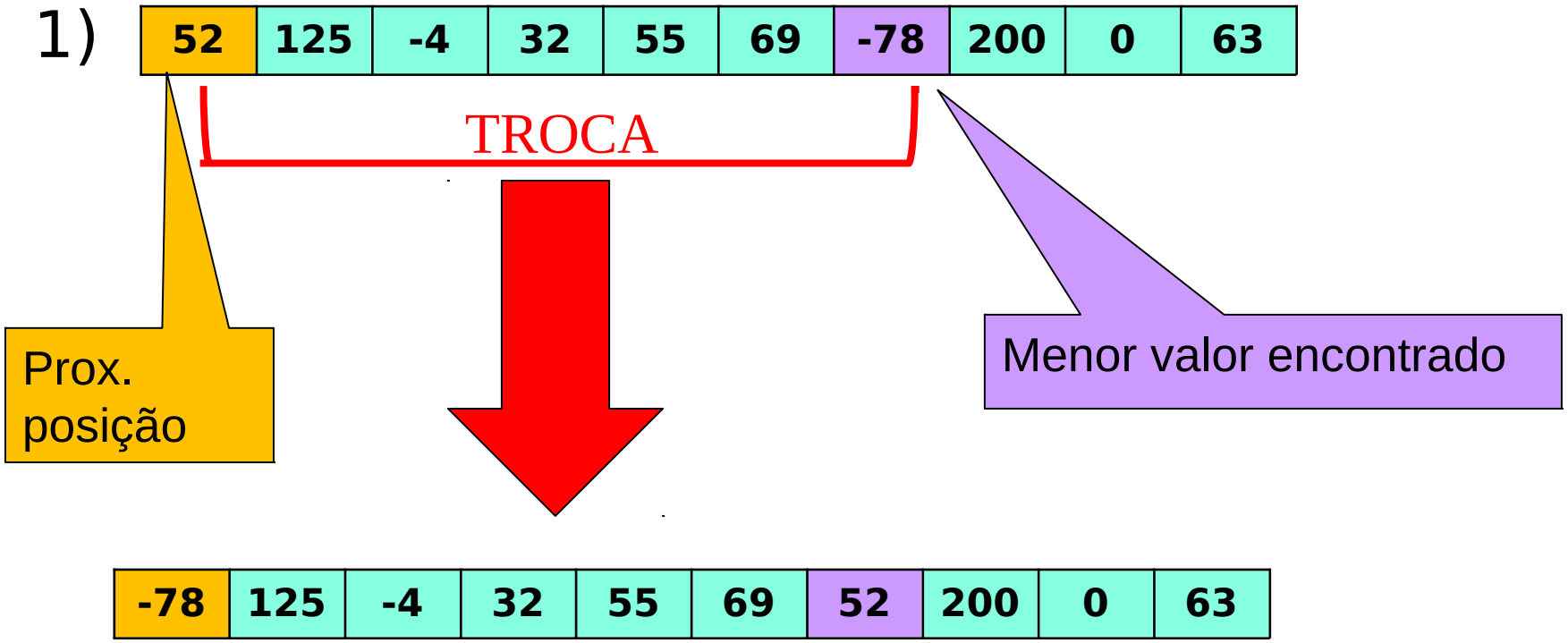
1)



# Primeiro método: seleção direta

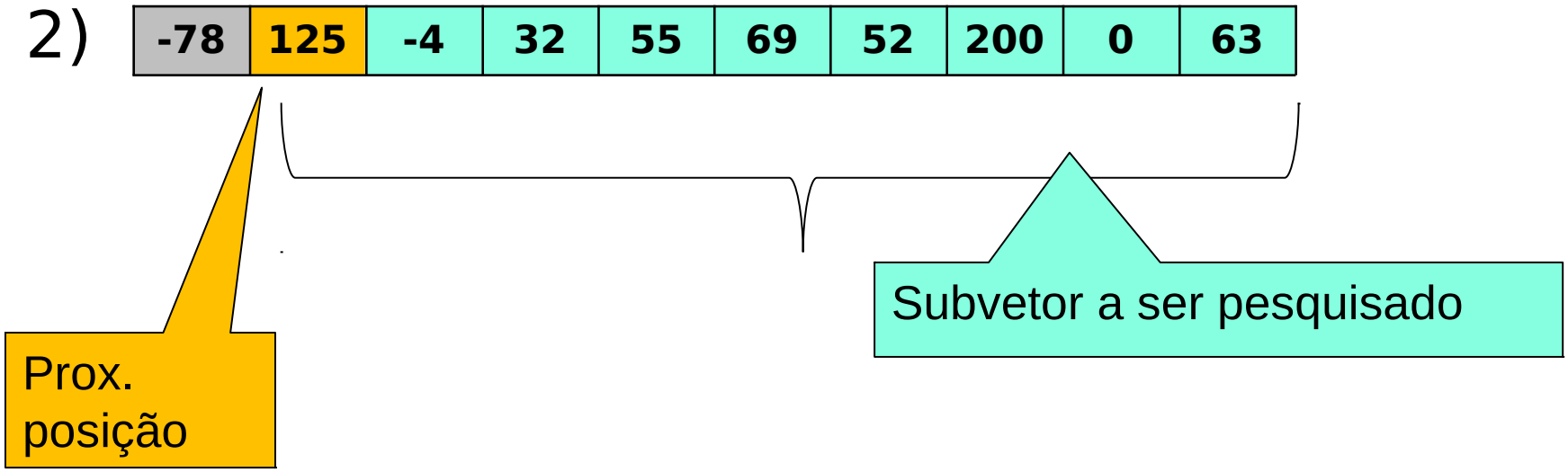


# Primeiro método: seleção direta



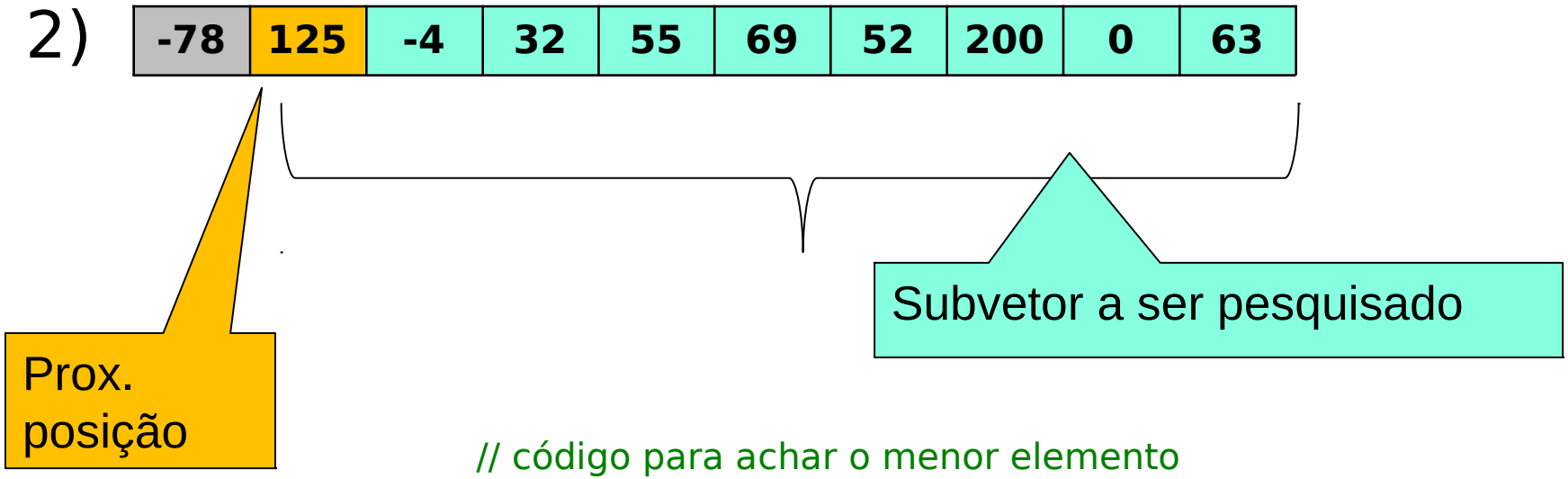
# Primeiro método: seleção direta

2)



# Primeiro método: seleção direta

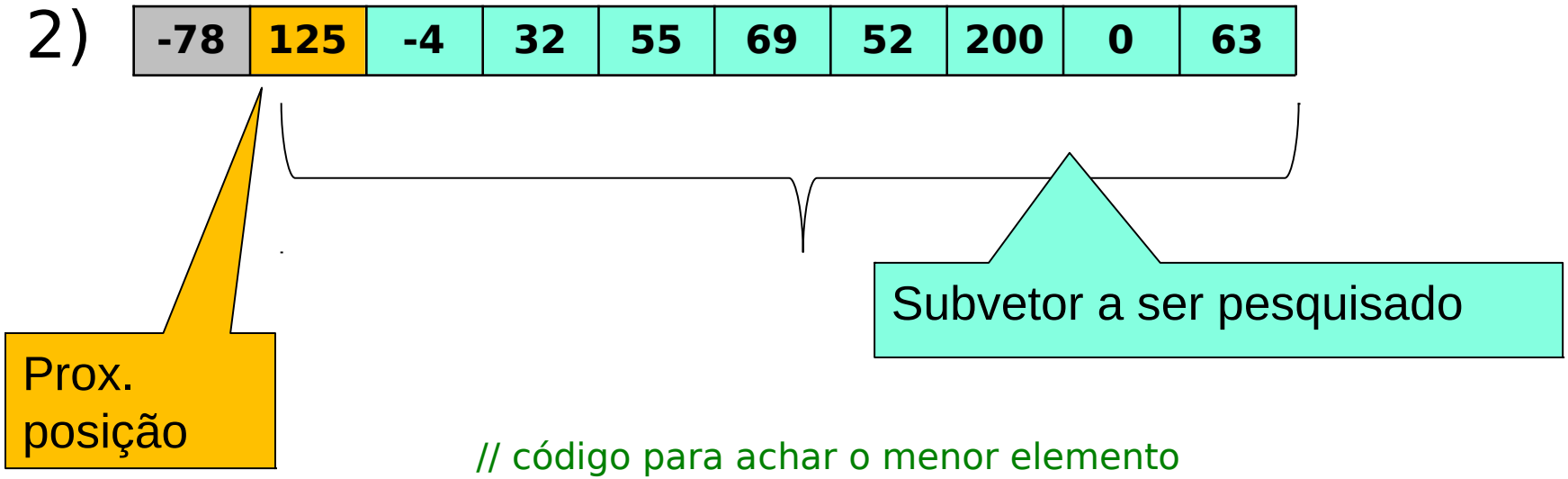
2)




```
// código para achar o menor elemento
int iMenor = ?; // posição inicial do vetor que está
    buscando
for (i=?; i < N; i++){
        if (num[i] < num[iMenor])
                iMenor = i;
}
```

# Primeiro método: seleção direta

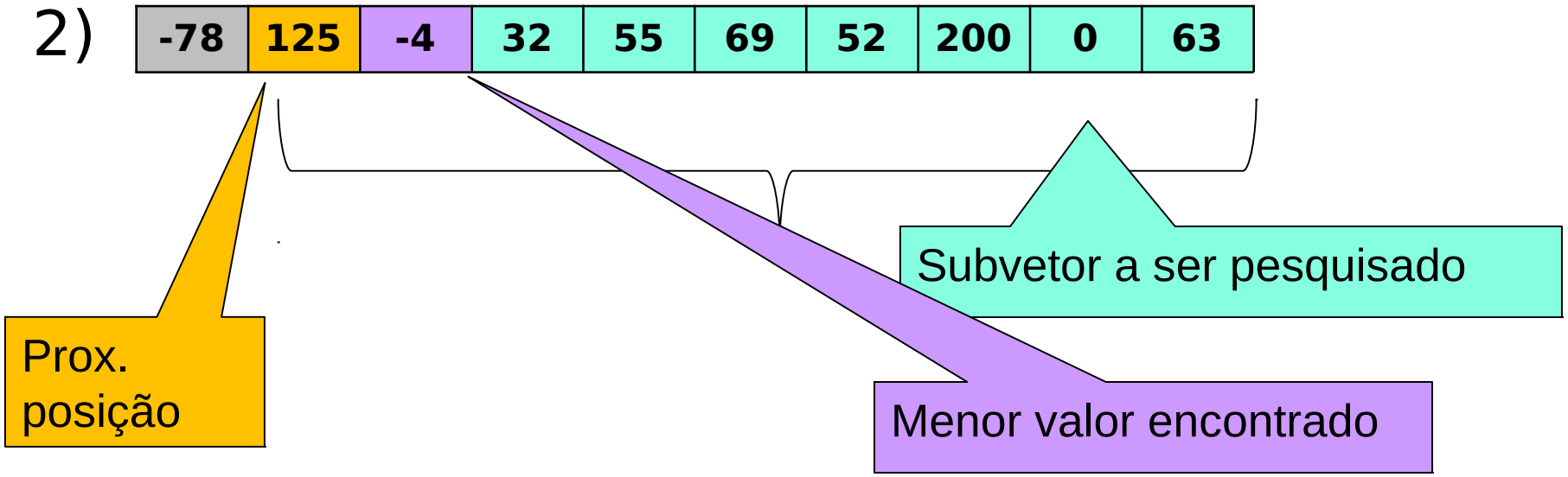
2)



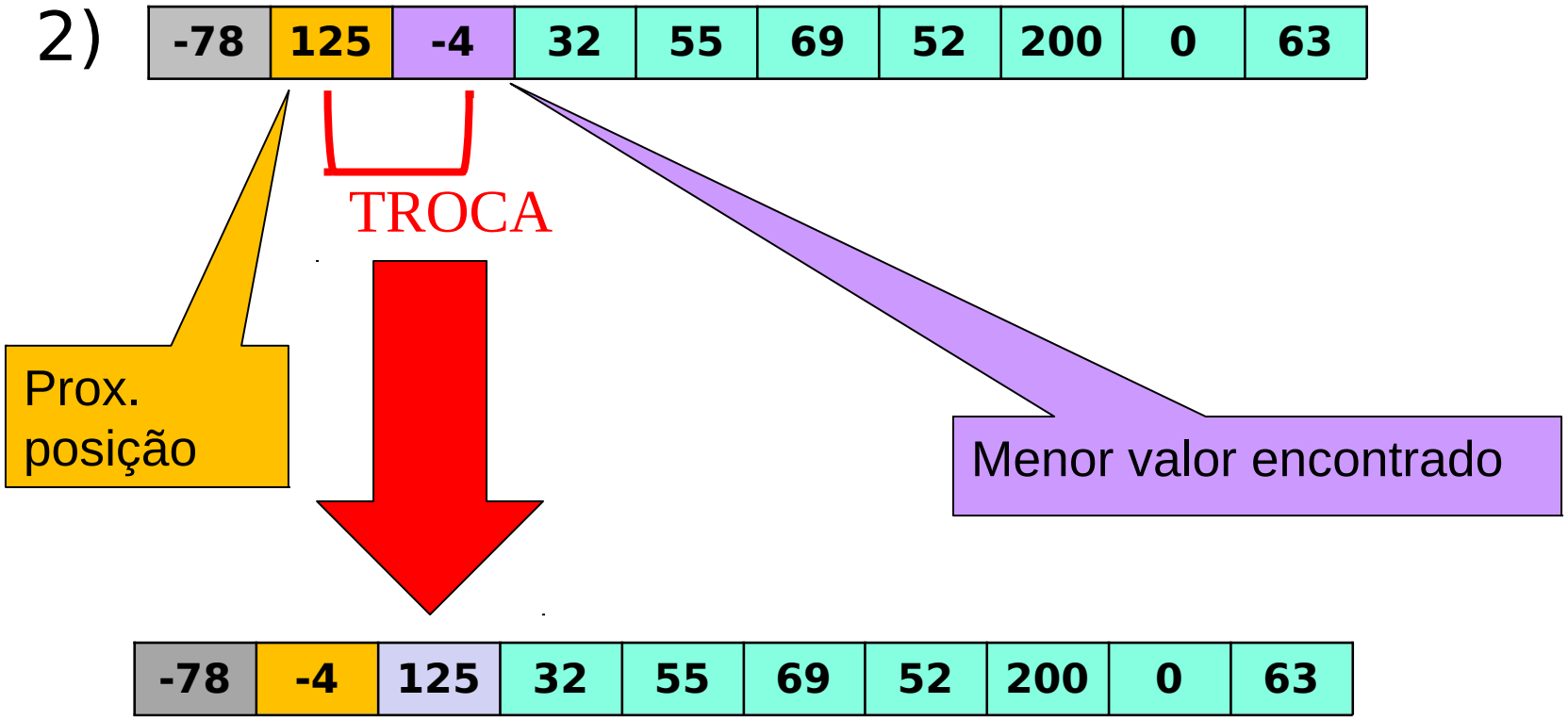

```
// código para achar o menor elemento
int iMenor = 1; // posição inicial do vetor que está
    buscando
for (i=2; i < N; i++){
        if (num[i] < num[iMenor])
                iMenor = i;
}
```

# Primeiro método: seleção direta

2)

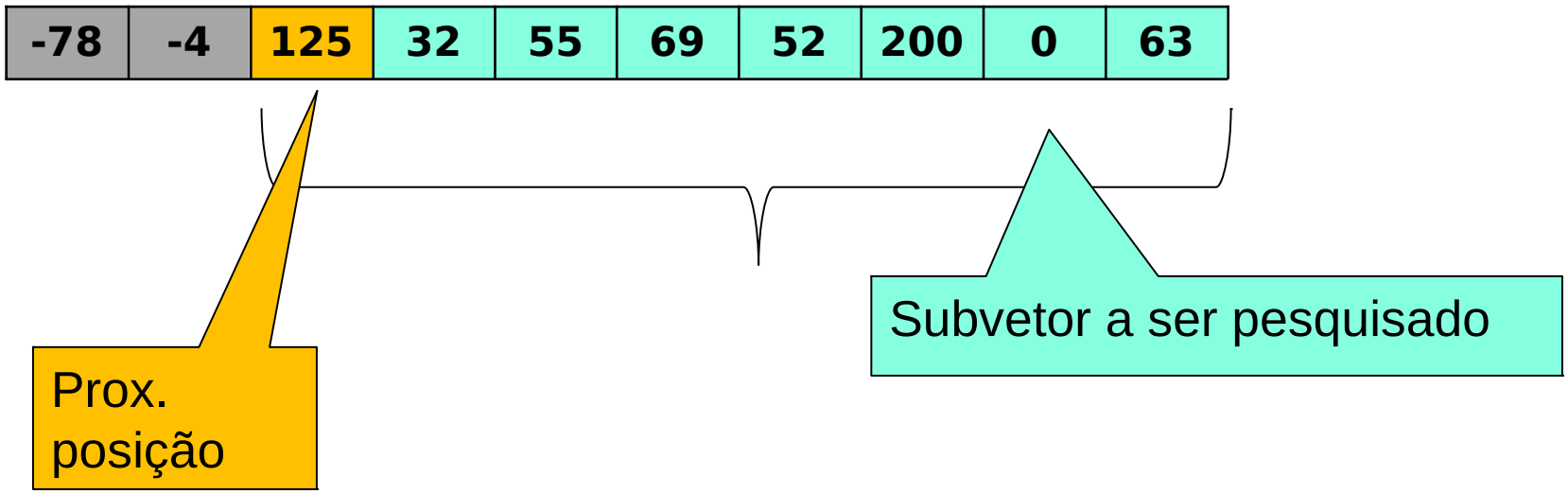


# Primeiro método: seleção direta



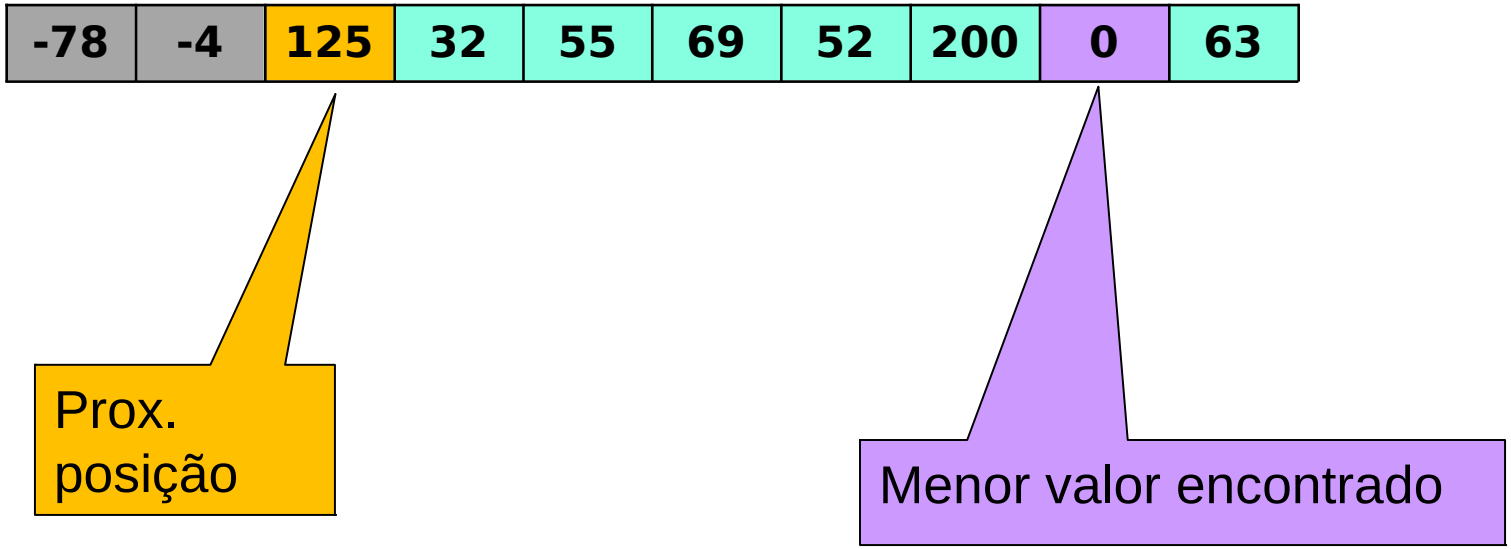
# Primeiro método: seleção direta

3)



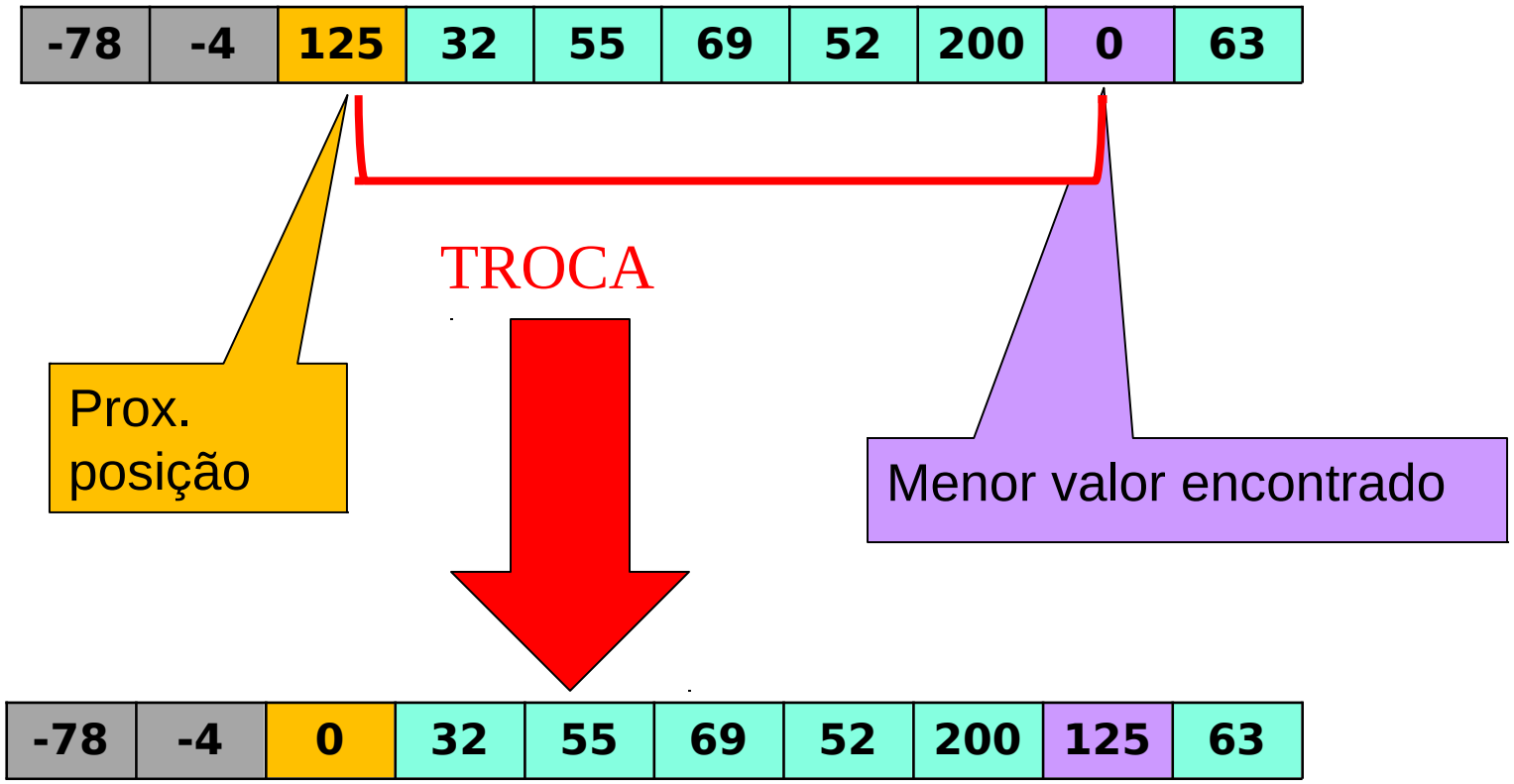
# Primeiro método: seleção direta

3)



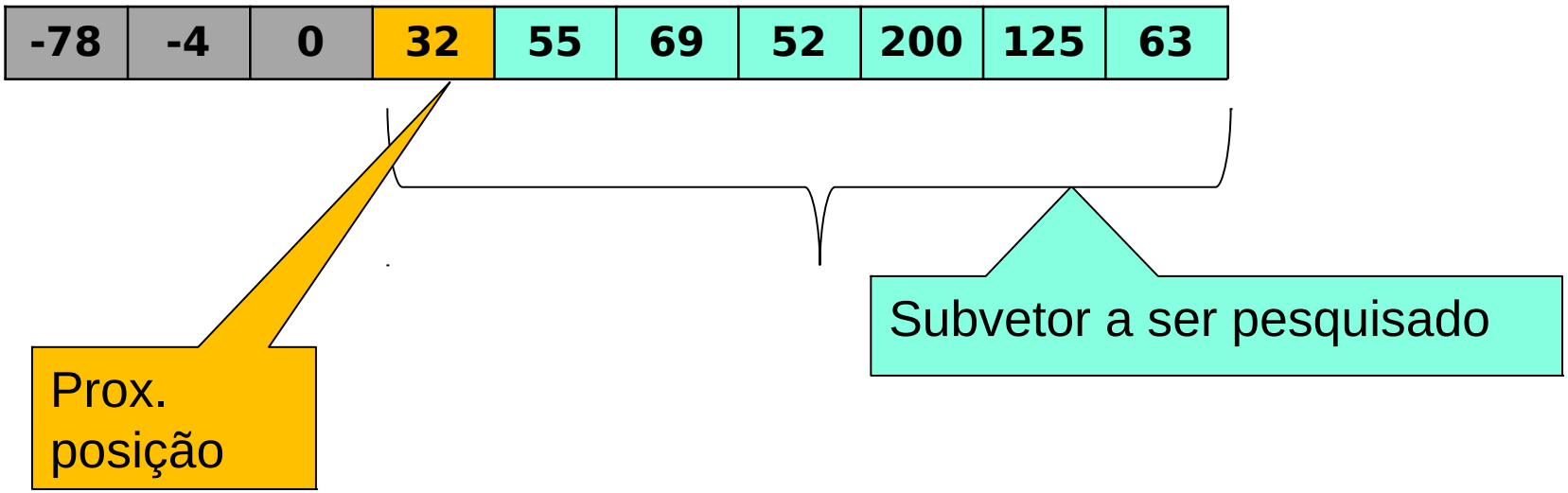
# Primeiro método: seleção direta

3)



# Primeiro método: seleção direta

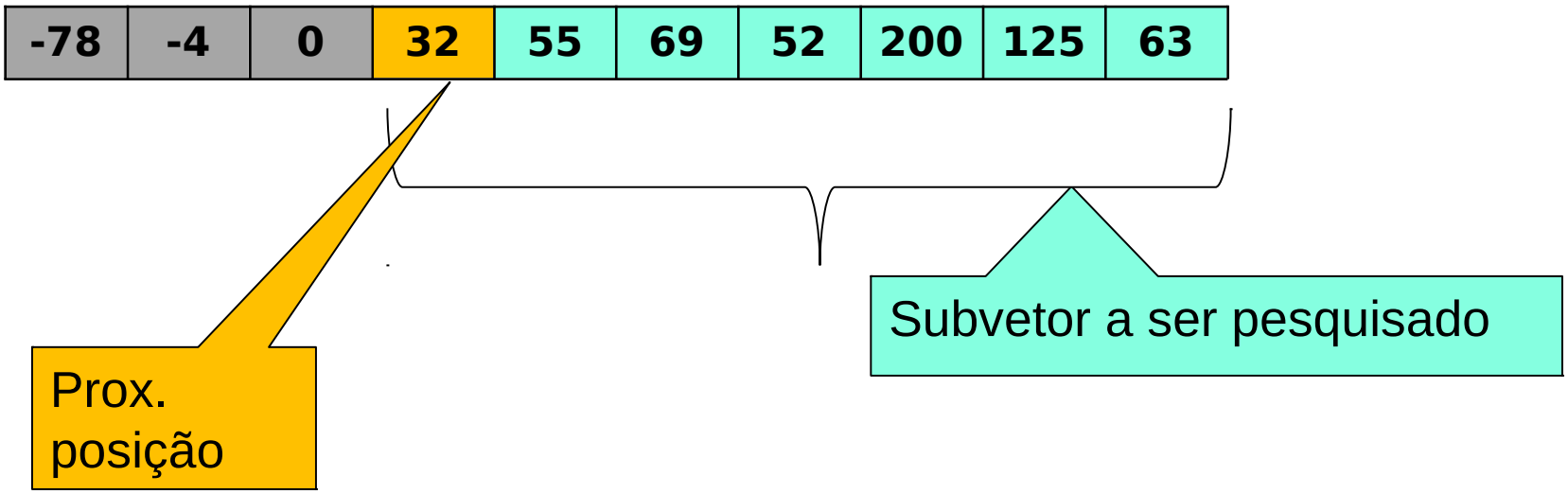
4)




**E assim por diante...**

# Primeiro método: seleção direta

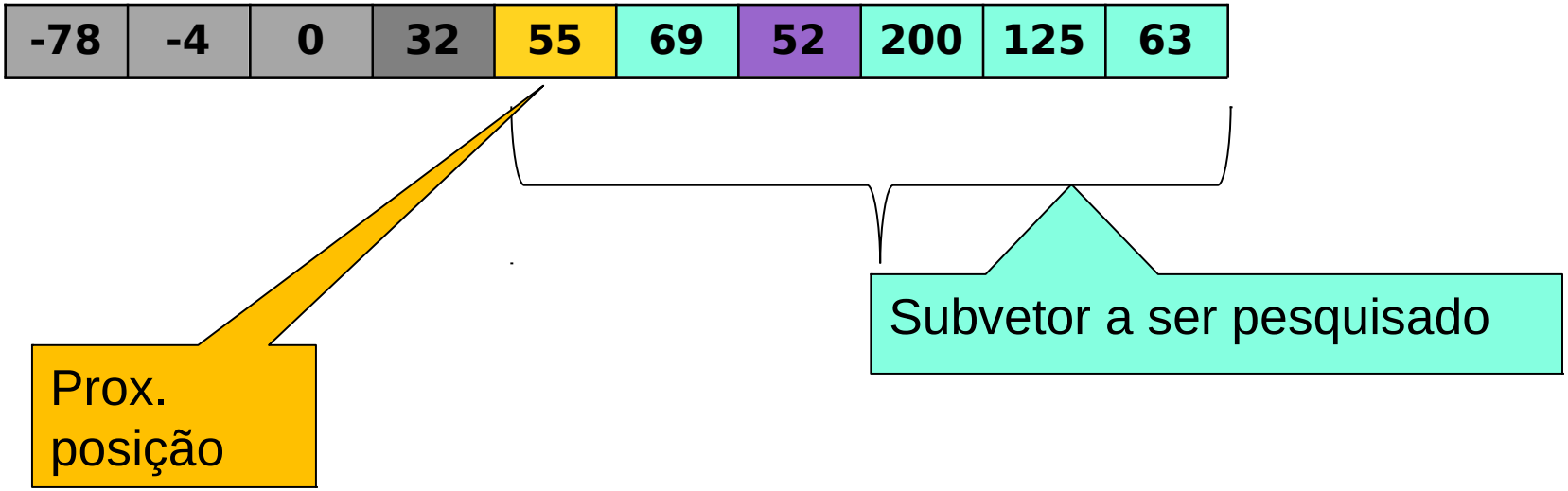
4)




**E assim por diante...**
**ATÉ QUANDO?**

# Primeiro método: seleção direta

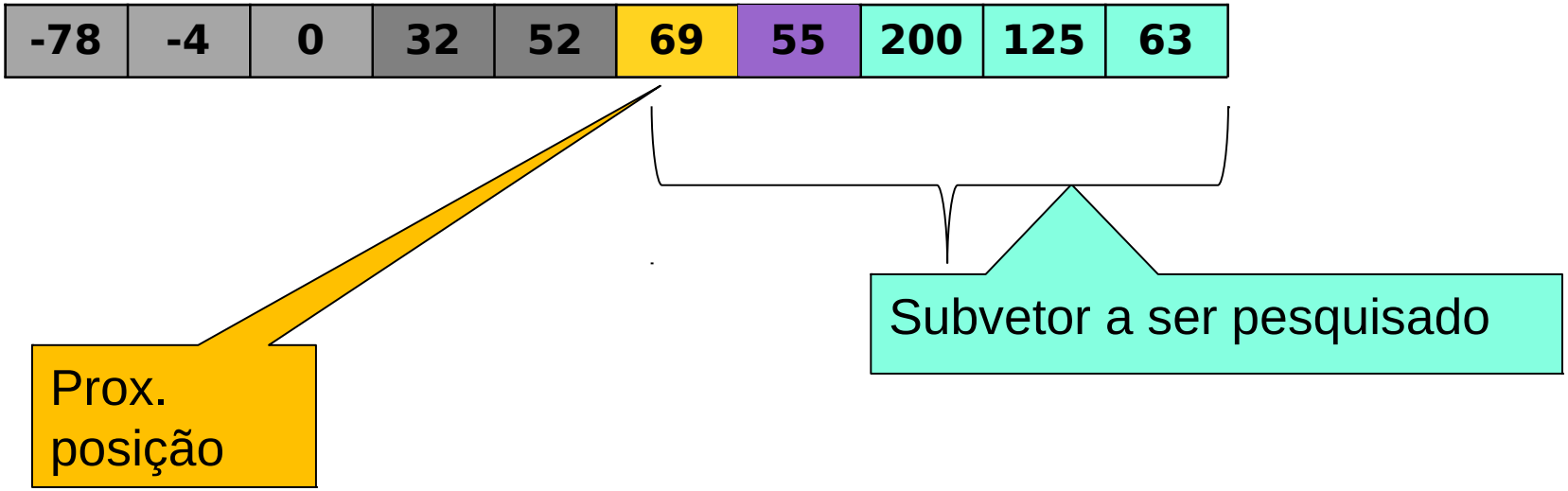
4)




**E assim por diante...**
**ATÉ QUANDO?**

# Primeiro método: seleção direta

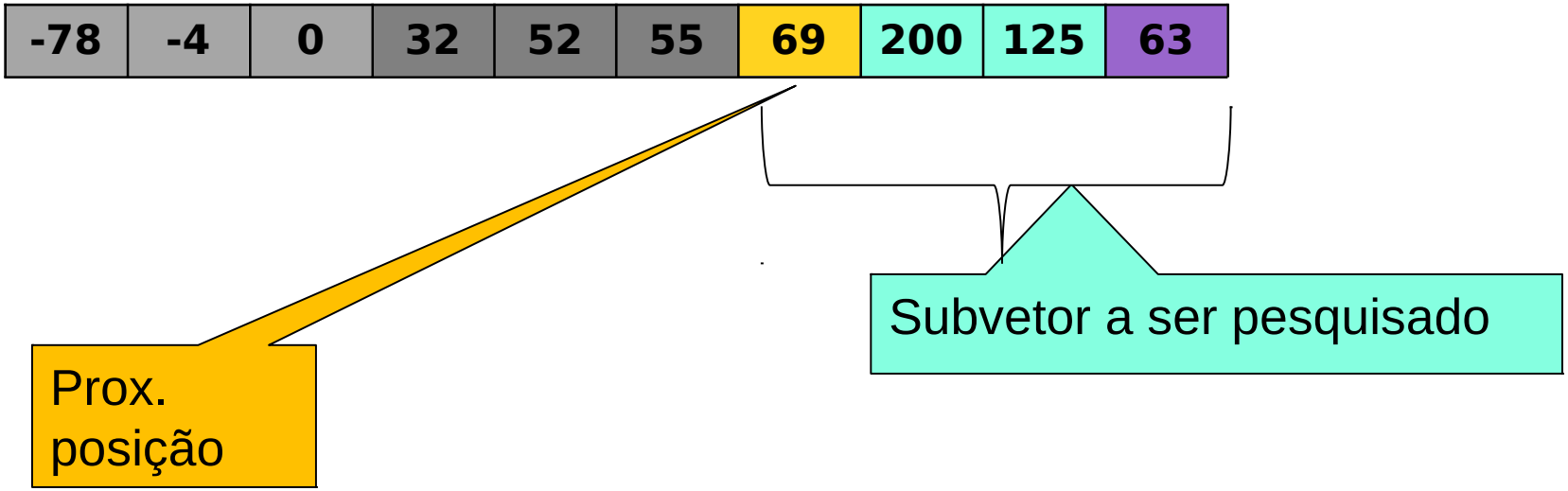
4)




**E assim por diante...**
**ATÉ QUANDO?**

# Primeiro método: seleção direta

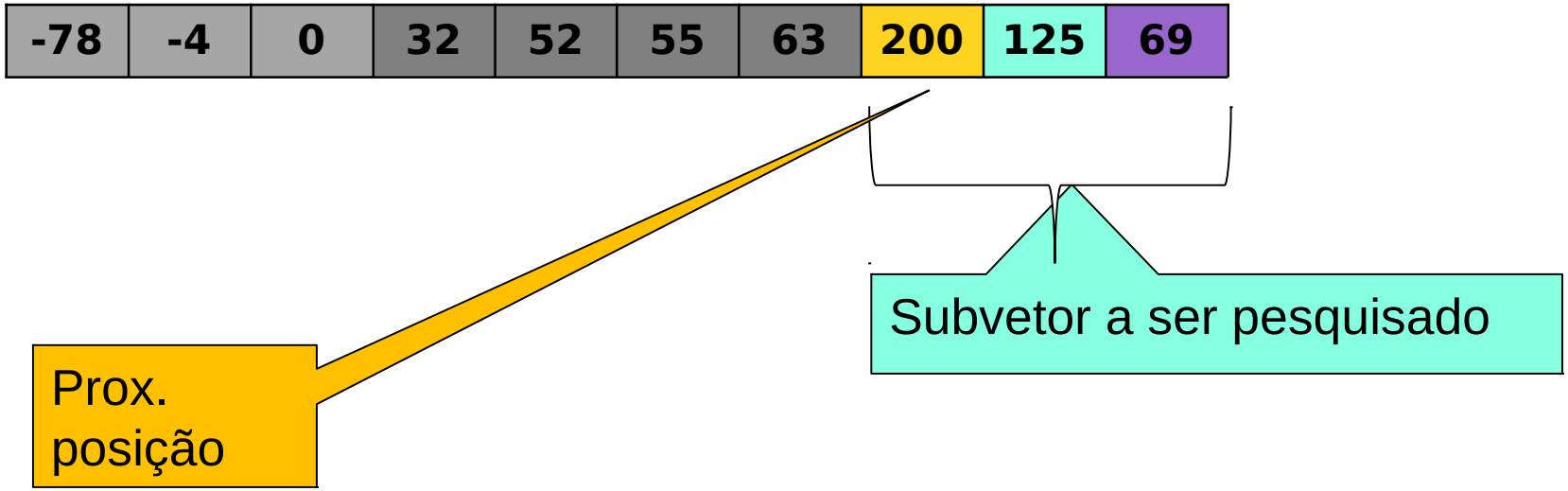
4)




**E assim por diante...**
**ATÉ QUANDO?**

# Primeiro método: seleção direta

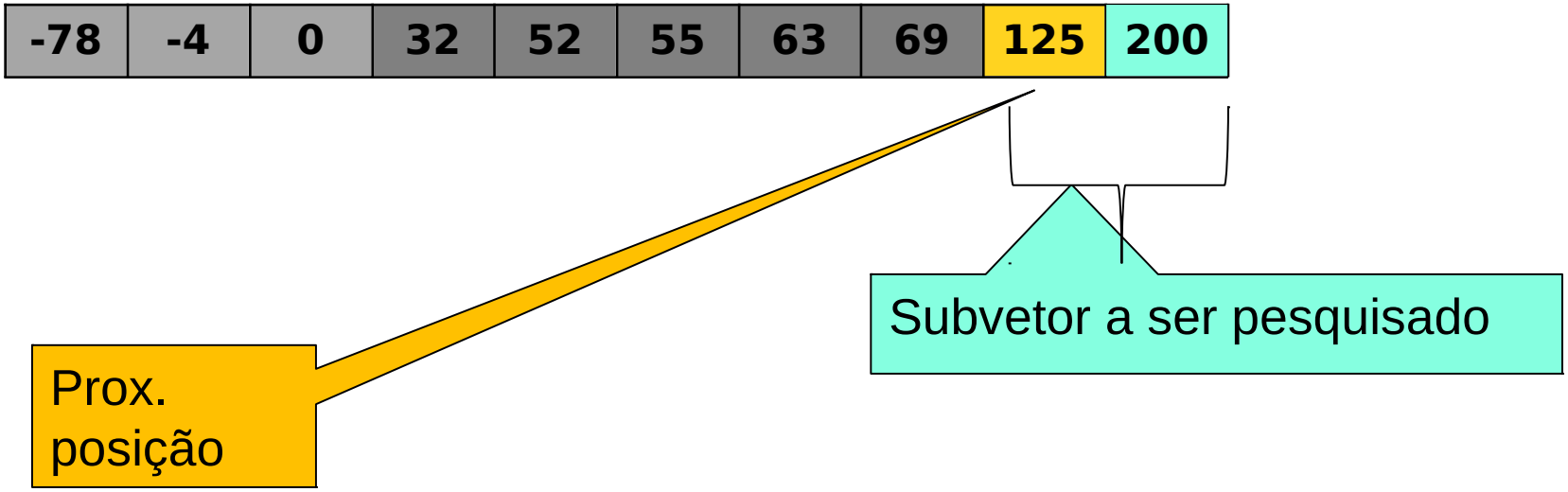
4)




**E assim por diante...**
**ATÉ QUANDO?**

# Primeiro método: seleção direta

4)




**E assim por diante...**
**ATÉ QUANDO?**

# Primeiro método: seleção direta

4)


?

Subvetor a ser pesquisado

Prox. posição

**E assim por diante...**
**ATÉ QUANDO?**

# Primeiro método: seleção direta

4)


?

Subvetor a ser pesquisado

Prox. posição

**E assim por diante...**
**PARO QUANDO PRÓXIMA POSIÇÃO = N - 1**

# Primeiro método: seleção direta

Algoritmo:

- para cada passo do vetor original, onde a próxima posição a ser preenchida vai de 0 a N-2
  - Percorre subvetor (da próxima posição a ser preenchida até tamanho do vetor -1) para encontrar menor valor no subvetor
  - Troca com o elemento da próxima posição a ser preenchida

```
void selecaoDireta(int numeros[], int N)
 {









 }
```

```
void selecaoDireta(int numeros[], int N)
 {
   int prox;
   for (prox=0; prox <  N -1; prox++) // percorre vetor principal
   {




   }
 }
```

1) Próxima posição a ser preenchida é controlada pelo for

```
void selecaoDireta(int numeros[], int N)
 {
  int prox, iMenor, i;
  for (prox=0; prox < N -1; prox++) // percorre vetor principal
  {
   // o primeiro elemento do vetor é o menor elemento inicialmente
   iMenor = prox;

   // percorre subvetor procurando o menor elemento
   for (i = prox + 1; i < N; i++)
   {
    if (numeros[i] < numeros[iMenor])
      iMenor = i;
   }




  }
 }
```

2) Percorrer subvetor para encontrar menor valor

```
void selecaoDireta(int numeros[], int N)
 {
   int prox, iMenor, i;
   for (prox=0; prox < N -1; prox++) // percorre vetor principal
   {
     // o primeiro elemento do vetor é o menor elemento inicialmente
     iMenor = prox;

     // percorre subvetor procurando o menor elemento
     for (i = prox + 1; i < N; i++)
     {
       if (numeros[i] < numeros[iMenor])
          iMenor = i;
     }

     // executa a troca da próxima posição com o menor elemento encontrado no
subvetor
     int temp = numeros[prox];
     numeros[prox] = numeros[iMenor];
     numeros[iMenor] = temp;
   }
 }
```

3) Troca o menor com a próxima posição a ser preenchida...

# Segundo método: inserção direta

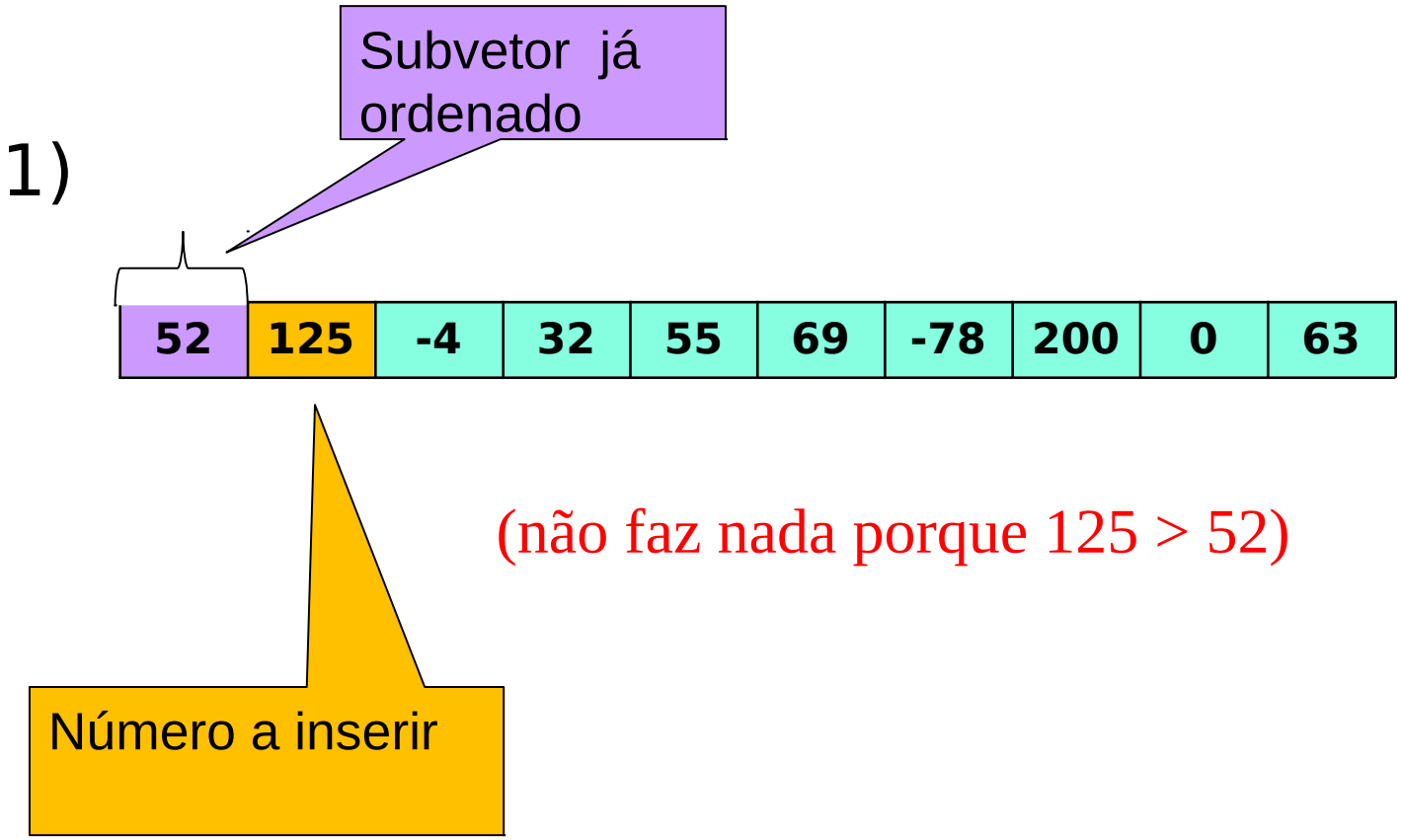
- Como você ordena as cartas do baralho, se pegá-las uma de cada vez?

# Segundo método: inserção direta

- Percorre o array e, em cada passo:
  - Aumenta a parte ordenada do array em uma posição, inserindo um novo elemento na posição correta e deslocando os demais para a direita

# Segundo método: inserção direta

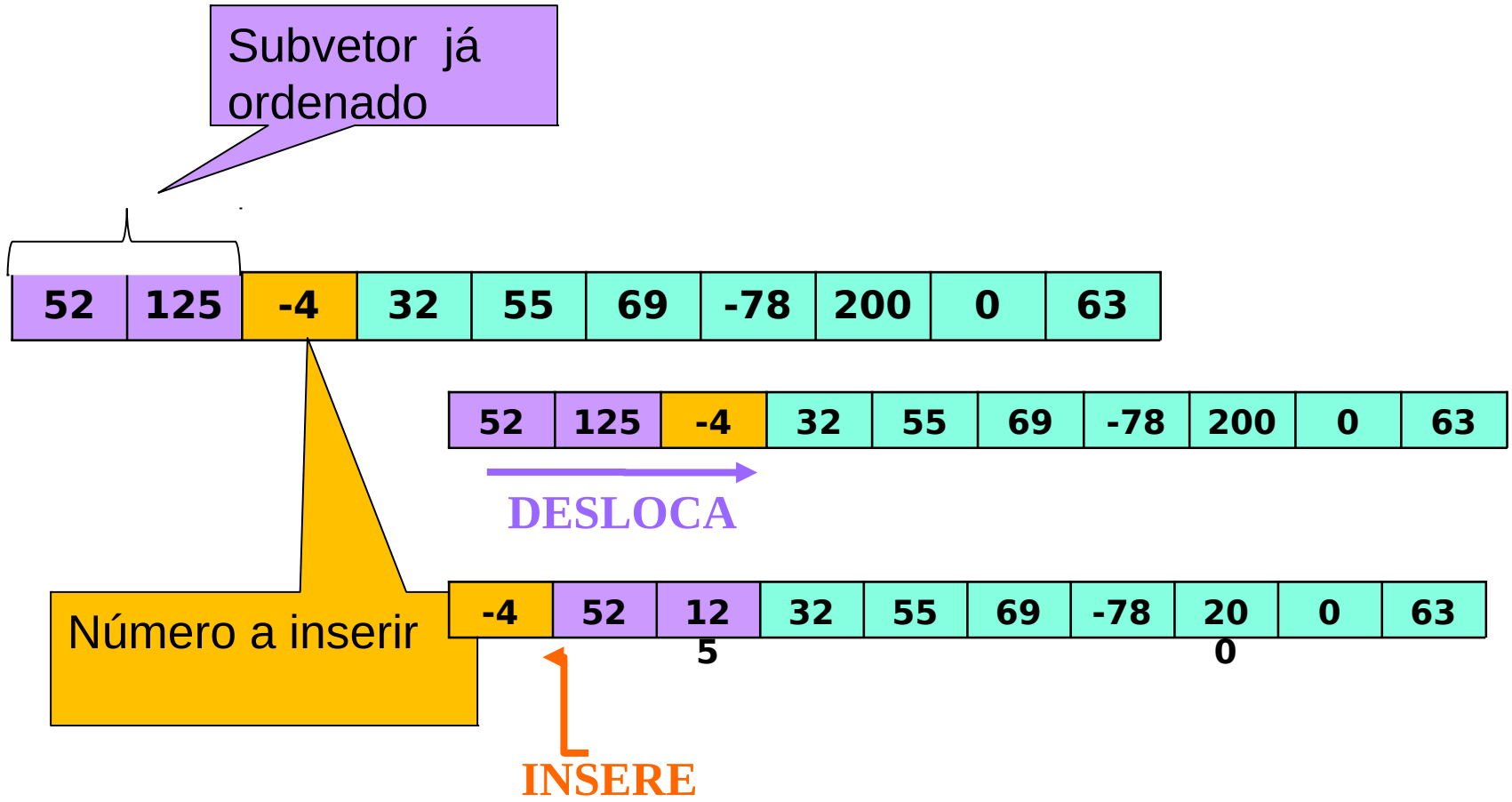
1)




(não faz nada porque 125 > 52)

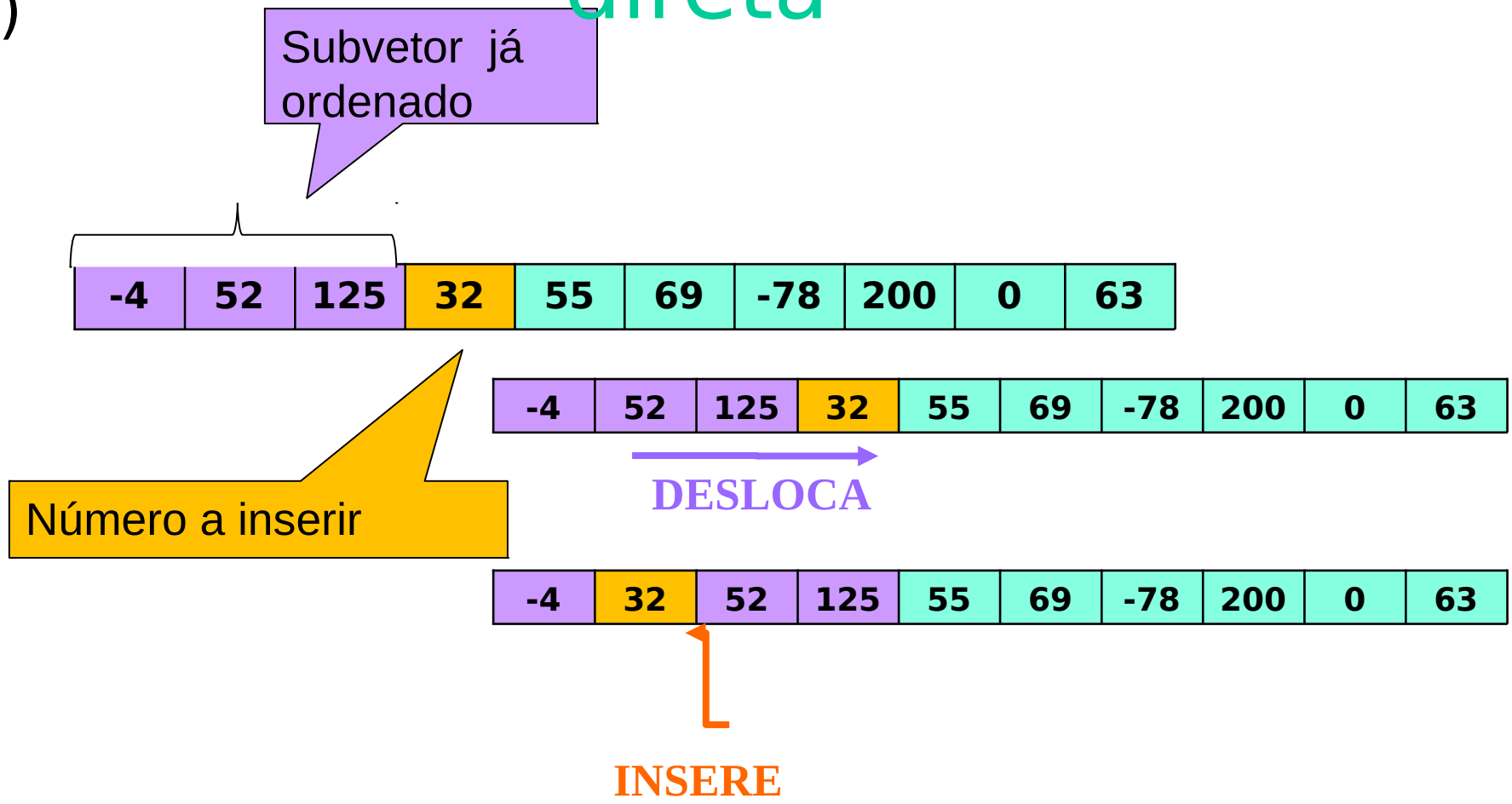
# Segundo método: inserção direta

2)



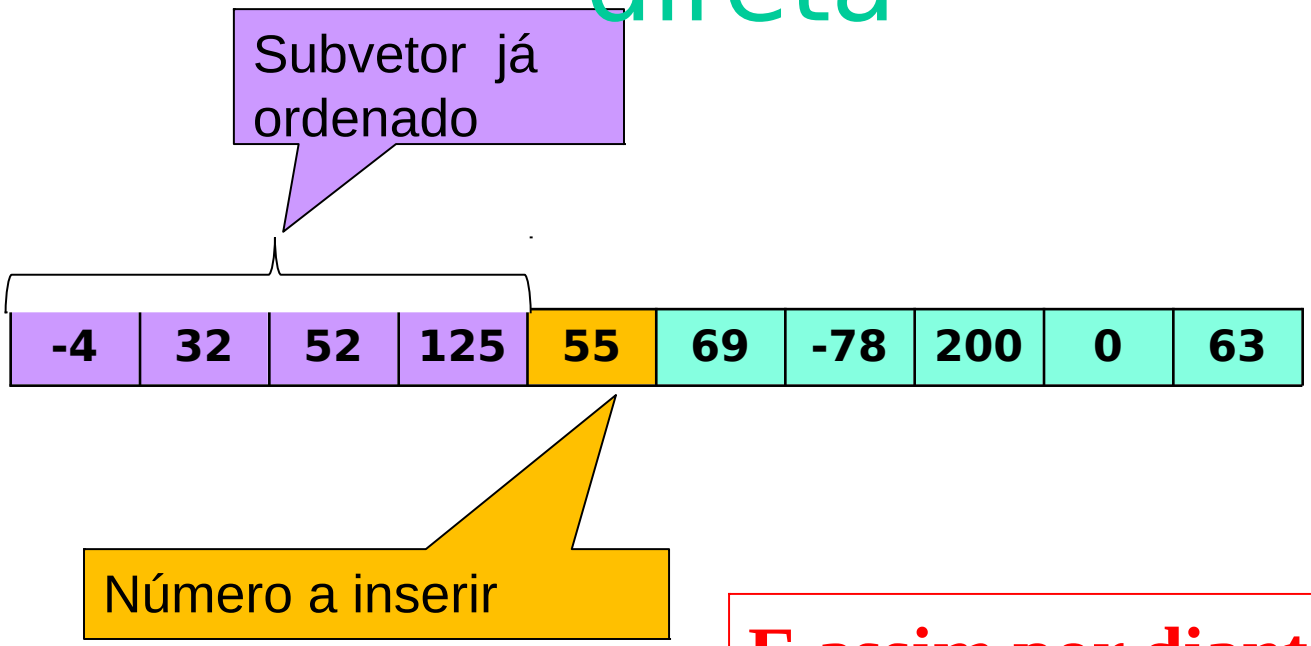
# Segundo método: inserção direta

3)



# Segundo método: inserção direta

3)




**E assim por diante...**

# Segundo método: inserção direta

Algoritmo:

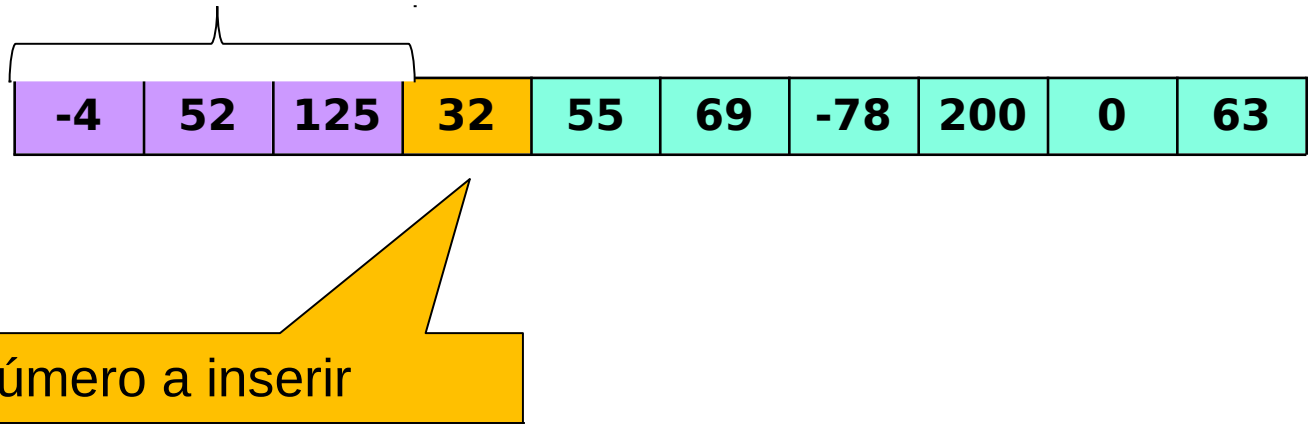
- para cada passo do vetor original, a partir da posição 1 até o final
  1. Desloca vetor ordenado para direita (uma posição por vez) até encontrar local adequado para o número a ser inserido
  2. Insere número na posição correta

```
void insercaoDireta(int numeros[], int N)
 {
```

```
void insercaoDireta(int numeros[], int N)
 {
  int ivet, numeroAInserir;
   // cada passo considera que o array à esquerda de i está ordenado
   for (ivet=1; ivet < N; ivet++) // percorre vetor principal
   {

     // número que será encaixado no vetor ordenado
     numeroAInserir = numeros[ivet];

   }
 }
```

Para cada posição a inserir
("do monte de cartas")

```
void insercaoDireta(int numeros[], int N)
 {
   int ivet, numeroAInserir;
   // cada passo considera que o array à esquerda de i está ordenado
   for (ivet=1; ivet < N; ivet++) // percorre vetor principal
   {
     // tenta inserir mais um número na porção inicial do array que
     // já está ordenado empurrando para a direita todos os elementos
     // maiores do que numeroAInserir

     // número que será encaixado no vetor ordenado
     numeroAInserir = numeros[ivet];
```



```
void insercaoDireta(int numeros[], int N)
 {
   int ivet, numeroAInserir, isubv;
   // cada passo considera que o array à esquerda de i está ordenado
   for (ivet=1; ivet < N; ivet++) // percorre vetor principal
   {
     // tenta inserir mais um número na porção inicial do array que
     // já está ordenado empurrando para a direita todos os elementos
     // maiores do que numeroAInserir

     // número que será encaixado no vetor ordenado
     numeroAInserir = numeros[ivet];
     isubv = ivet;
     while ((isubv > 0) && (numeros [isubv -1] > numeroAInserir))
     // desloca vetor ordenado até encontrar posição para o número a inserir
     {
          numeros[isubv] = numeros[isubv - 1];
          isubv--;
     }


   }
 }
```

Desloca vetor para direita

```
void insercaoDireta(int numeros[], int N)
 {
   int ivet, numeroAInserir, isubv;
   // cada passo considera que o array à esquerda de i está ordenado
   for (ivet=1; ivet < N; ivet++) // percorre vetor principal
   {
     // tenta inserir mais um número na porção inicial do array que
     // já está ordenado empurrando para a direita todos os elementos
     // maiores do que numeroAInserir

     // número que será encaixado no vetor ordenado
     numeroAInserir = numeros[ivet];
     isubv = ivet;
     while ((isubv > 0) && (numeros [isubv -1] > numeroAInserir))
     // desloca vetor ordenado até encontrar posição para o número a inserir
     {
          numeros[isubv] = numeros[isubv - 1];
          isubv--;
     }
     numeros[isubv] = ?;
   }
 }
```

Insere número no local correto

```
void insercaoDireta(int numeros[], int N)
 {
   int ivet, numeroAInserir, isubv;
   // cada passo considera que o array à esquerda de i está ordenado
   for (ivet=1; ivet < N; ivet++) // percorre vetor principal
   {
     // tenta inserir mais um número na porção inicial do array que
     // já está ordenado empurrando para a direita todos os elementos
     // maiores do que numeroAInserir

     // número que será encaixado no vetor ordenado
     numeroAInserir = numeros[ivet];
     isubv = ivet;
     while ((isubv > 0) && (numeros [isubv -1] > numeroAInserir))
     // desloca vetor ordenado até encontrar posição para o número a inserir
     {
          numeros[isubv] = numeros[isubv - 1];
          isubv--;
     }
     numeros[isubv] = numeroAInserir;
   }
 }
```

Insere número no local correto

# Terceiro método: bolha

- Percorre o array e, em cada passo:
  - Testa se a sequência está ordenada
  - Troca o par que não estiver ordenado

1)

| 52 | 125 | -4 | 32 | 55 | 69 | -78 | 200 | 0 | 63 |
|---|---|---|---|---|---|---|---|---|---|

não troca troca

| 52 | -4 | 125 | 32 | 55 | 69 | -78 | 200 | 0 | 63 |
|---|---|---|---|---|---|---|---|---|---|

troca

| 52 | -4 | 32 | 125 | 55 | 69 | -78 | 200 | 0 | 63 |
|---|---|---|---|---|---|---|---|---|---|

troca

| 52 | -4 | 32 | 55 | 125 | 69 | -78 | 200 | 0 | 63 |
|---|---|---|---|---|---|---|---|---|---|

troca

| 52 | -4 | 32 | 55 | 69 | 125 | -78 | 200 | 0 | 63 |
|---|---|---|---|---|---|---|---|---|---|

troca

| 52 | -4 | 32 | 55 | 69 | -78 | 125 | 200 | 0 | 63 |
|---|---|---|---|---|---|---|---|---|---|

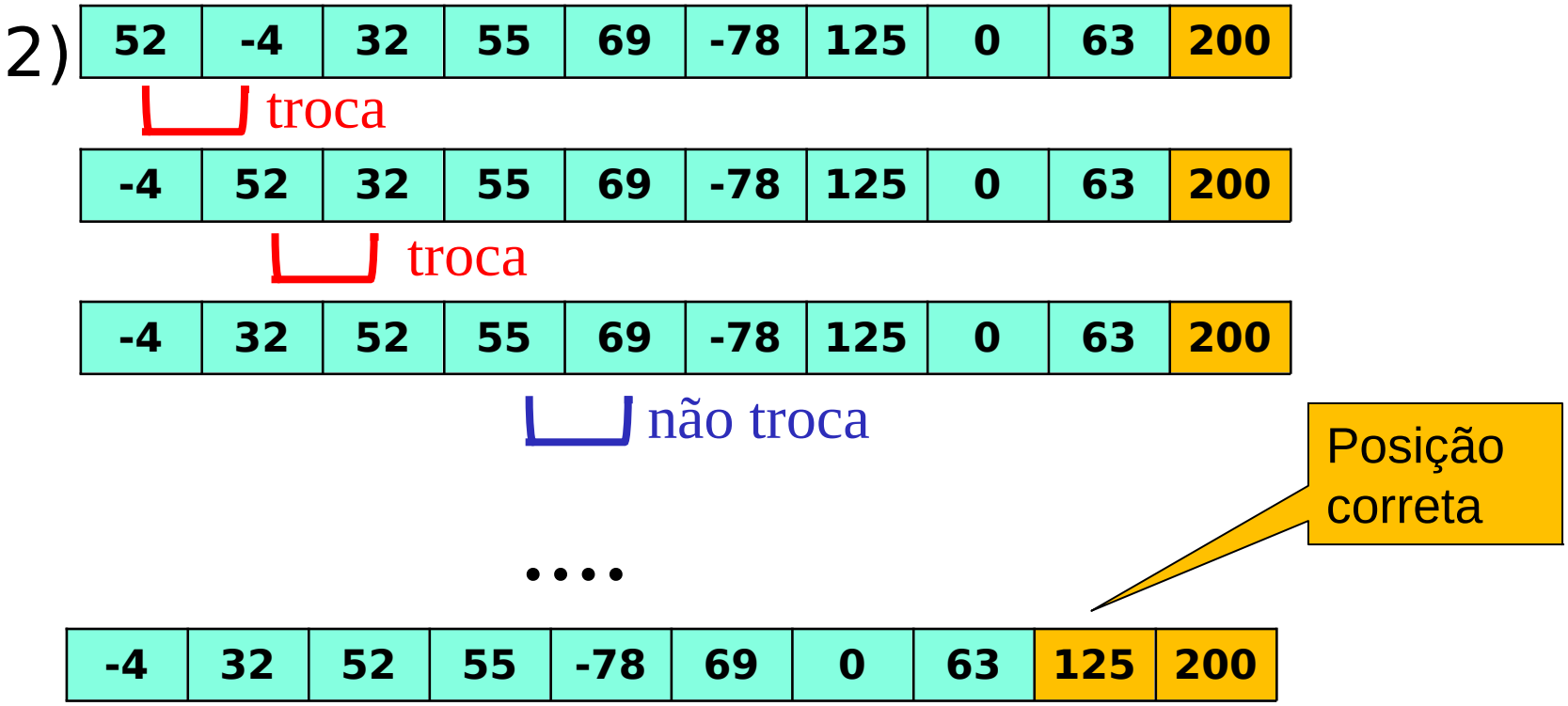
não troca troca

| 52 | -4 | 32 | 55 | 69 | -78 | 125 | 0 | 200 | 63 |
|---|---|---|---|---|---|---|---|---|---|

troca

Posição correta

| 52 | -4 | 32 | 55 | 69 | -78 | 125 | 0 | 63 | 200 |
|---|---|---|---|---|---|---|---|---|---|


2)
52 -4 32 55 69 -78 125 0 63 200
troca
-4 52 32 55 69 -78 125 0 63 200
troca
-4 32 52 55 69 -78 125 0 63 200
não troca
Posição correta
....
-4 32 52 55 -78 69 0 63 125 200

2)

| 52 | -4 | 32 | 55 | 69 | -78 | 125 | 0 | 63 | 200 |
|---|---|---|---|---|---|---|---|---|---|

troca

| -4 | 52 | 32 | 55 | 69 | -78 | 125 | 0 | 63 | 200 |
|---|---|---|---|---|---|---|---|---|---|

troca

| -4 | 32 | 52 | 55 | 69 | -78 | 125 | 0 | 63 | 200 |
|---|---|---|---|---|---|---|---|---|---|

não troca

...

Posição correta

| -4 | 32 | 52 | 55 | -78 | 69 | 0 | 63 | 125 | 200 |
|---|---|---|---|---|---|---|---|---|---|

**E assim por diante...**

# Terceiro método: bolha

Algoritmo:

- Quantas vezes?
  - Percorre o vetor desde o início até onde já estiver ordenado (ivet) e vai trocando o par de lugar quando posicão [i-1] > posição [i]

# Terceiro método: bolha

Algoritmo:

- para cada posição (do vetor original) ivet onde começa o subvetor ordenado (a partir da posição ? até ?)
  - Percorre o vetor desde o início até onde já estiver ordenado (ivet) e vai trocando o par de lugar quando posicão [i-1] > posição [i]

# Terceiro método: bolha

Algoritmo:

- para cada posição (do vetor original) ivet onde começa o subvetor ordenado (a partir da a partir da posição tamanho – 1 até 1)
  - Percorre o vetor desde o início até onde já estiver ordenado (ivet) e vai trocando o par de lugar quando posicão [i-1] > posição [i]

# Terceiro método: bolha

```
void bolha(int numeros[], int N)
  {




  }
```

# Terceiro método: bolha

```
void bolha(int numeros[], int N)
  {
    int ivet;
    // percorre vetor principal do fim para o início
    for (ivet = N - 1; ivet > 0; ivet--)
    {
          // a cada passagem o maior elemento é descolado para a sua posição correta









     }
  }
```

# Terceiro método: bolha

```
void bolha(int numeros[], int N)
  {
    int ivet, isubv;
    // percorre vetor principal do fim para o início
    for (ivet = N - 1; ivet > 0; ivet--)
    {
          // a cada passagem o maior elemento é descolado para a sua posição correta

          for (isubv = 0; isubv < ?; isubv++)









     }
  }
```

# Terceiro método: bolha

```
void bolha(int numeros[], int N)
  {
    int ivet, isubv;
    // percorre vetor principal do fim para o início
    for (ivet = N - 1; ivet > 0; ivet--)
    {
          // a cada passagem o maior elemento é descolado para a sua posição correta
          // não há necessidade de ir até o final
          for (isubv = 0; isubv < ivet; isubv++)
          // se ordem está errada para dois números consecutivos, troca os números









     }
  }
```

# Terceiro método: bolha

```
void bolha(int numeros[], int N)
  {
    int ivet, isubv, temp;
    // percorre vetor principal do fim para o início
    for (ivet = N - 1; ivet > 0; ivet--)
    {
          // a cada passagem o maior elemento é descolado para a sua posição correta
          // não há necessidade de ir até o final
          for (isubv = 0; isubv < ivet; isubv++)
          // se ordem está errada para dois números consecutivos, troca os números
             if (numeros[isubv ] > numeros[isubv+1])
             {
                   temp = numeros[isubv];
                   numeros [isubv] = numeros [isubv+1];
                   numeros [isubv+1] = temp;
             }
     }
  }
```

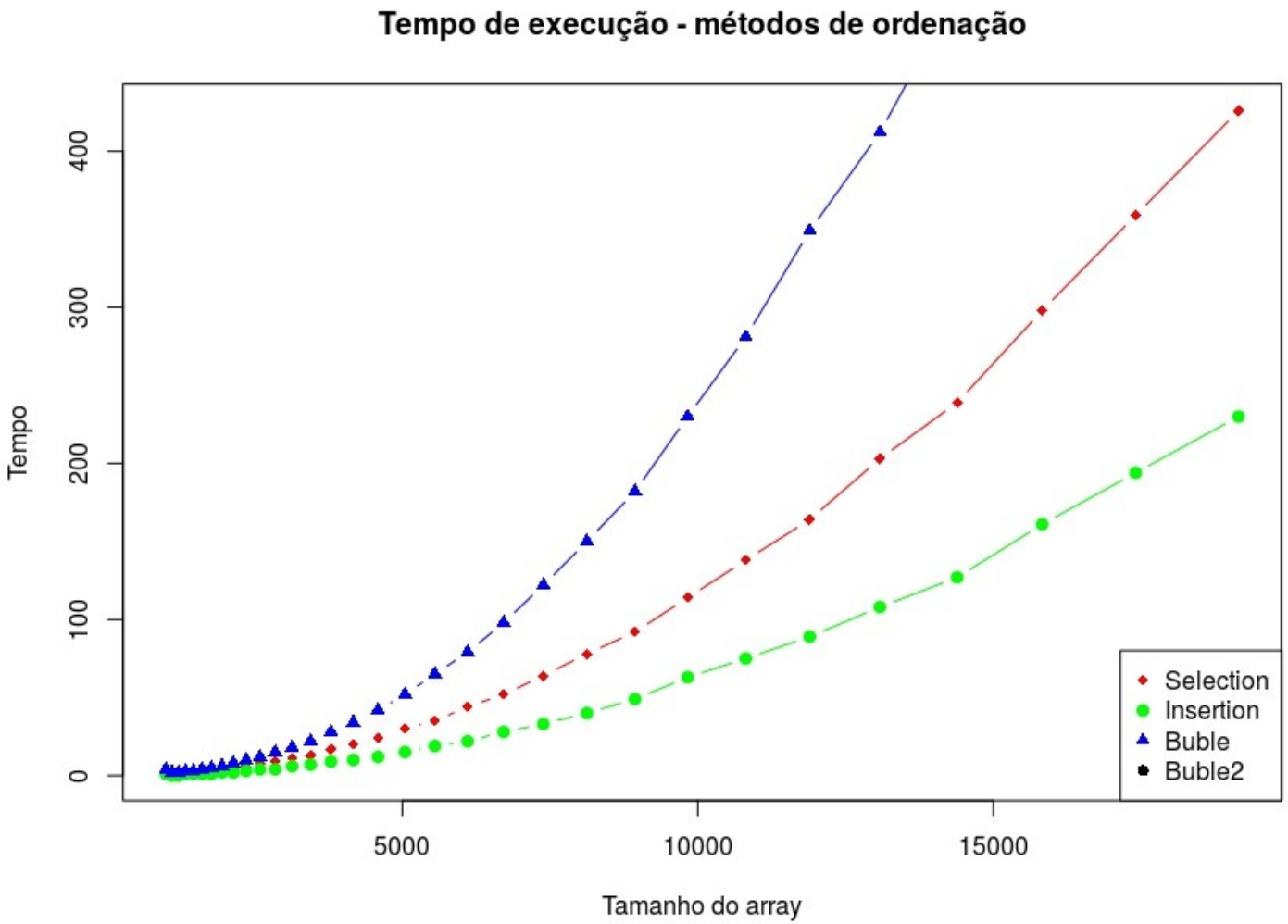
# Comparações entre os métodos

- Seleção e inserção direta:
  - garante que, no passo i, o subvetor de 0 a i está ordenado.
  - diferem em como fazem isso:
    - seleção troca 2 elementos
    - inserção desloca todo o sub-vetor à direita
- Bolha:
  - Garante que, no passo i, o subvetor de tam-1-i a tam-1 está ordenado
- Todos ordenam *in loco* (no próprio vetor, sem precisar de vetor auxiliar)

# Qual deles é melhor

- Mais rápido
- Menos memória
- Mais fácil de entender

# Comparação


Tempo de execução - métodos de ordenação
Tempo
0
100
200
300
400
5000
10000
15000
Tamanho do array
Selection
Insertion
Buble
Buble2

# Busca

# e

# Ordenação

Professores:
Fátima L. S. Nunes